

## Tutorial: Serial Communication in Matlab

**Objective:** This tutorial teaches you how to use Matlab serial objects to interact with external hardware.

**Disclaimer:** The tutorial is not a replacement for reading the Matlab Documentation on Serial Interfacing; nor does it cover all the concepts and implementation details of serial communication and the RS-232 protocol. The examples are solely geared toward the types of applications we see in our projects such as sending commands to control robots, motors, or reading sensors such as GPS, laser scanners, compasses, etc. Other types of equipment may require different techniques not covered here.

**Contact:**

Associate Professor Joel M. Esposito  
[esposito@usna.edu](mailto:esposito@usna.edu)  
Systems Engineering Department  
United States Naval Academy,  
Annapolis, MD 21402  
<http://www.usna.edu/Users/weapsys/esposito/>

## Hardware Devices

A lot of external hardware devices are designed to connect to a PC through a Serial Port. Examples in our department include Koala Robots, iCreate Robots, Scorbot, Robix Kits; and lots of sensors such as the Sick or Hokuyo Laser Scanners, the Northstar kits, GPS, Compasses, etc

Traditionally all PC's had a serial port in the back. However, now they are being replaced by USB ports. "Serial ports" take many forms. For example the Xbee modems create a wireless serial link. Many laptops don't have a serial port – just USB ports. USB to Serial Converters or Bluetooth Connections can function as "virtual serial ports", meaning that once set up correctly Matlab just veivs them as additional serial ports.

## Basic Concepts

- *Cabling:*
- *Serial Message:* You litterally send or recive data over this cable on a single pin as a *series* of bytes (1 byte = 8 bits or 0 – 255).
  - Example: [0] [12] [27] [42] [112]
- *Terminators:* Just as we use a period in English to dentote the end of a sentence, we use a "terminator" to indicate the end of a series of bytes that constitute a message.
  - The terminator can be anything the sender and receiver agree on but a "carrage return" (\r) is a common choice.
- *Buffer:* If you don't understand how a buffer works, you will never understand serial communication. Say a sensor is streaming back data to your program, more frequently than your program reads it. On your computer this data gets stored in something called a buffer, until you decide to read it. Think of a buffer as a list.
  - As new data values come in they get added to the bottom of the list (most recent data).
  - If your program reads a value from the buffer, it starts at the top of the list (oldest data). Once you read a byte of data, it is no longer in the buffer; and the data in the second position on the list moves up to the top position, etc.
  - The buffer has a finite length (you set it). This means there is a limit to how long the list can get. Once the buffer is totally full, what happens when the sensor tries to send new data to the buffer? The oldest data (top of the list) gets discarded forever, and all the entries move up, to make room on the bottom of the list for new data.
  - If you're smart about using the buffer, you can make sure you never miss any data. If your not smart about it, it is easy to loose data or use old data.

Ex: We create a buffer of length 5. Initially it is empty

1	
2	
3	
4	
5	

The sensor writes a value (10) to it

1	<b>10</b>
2	
3	
4	
5	

The sensor writes another value (6) to it, Note that the oldest data is in the first position and new data fills in the buffer from the bottom.

1	<b>10</b>
2	<b>6</b>
3	
4	
5	

Now we read a value from it into the variable x.

1	<b>6</b>
2	
3	
4	
5	

x= 10. Note that once we read the value it is no longer in the buffer! Also note that we read the top element which was the oldest data.

Here is another scenario. The buffer is full

1	<b>10</b>
2	<b>6</b>
3	<b>12</b>
4	<b>3</b>
5	<b>1</b>

Now the sensor writes a value to it (4)

1	<b>6</b>
2	<b>12</b>
3	<b>3</b>
4	<b>1</b>
5	<b>4</b>

The oldest data, in the top entry, (10) is discarded forever and all the entries shift up 1 spot, to make room for the new value (4).

- *Message Length and Check Sum:* It's very possible, especially when you use wireless serial connections that individual bytes in the message can get lost or garbled. There are a lot of complex schemes out there to check that this doesn't happen. Here are two common ones that show up in some serial command interfaces.
  - One of the bytes in the message might indicate the length of the message (the total number of bytes the message *should* contain).
  - Another byte might be a checksum. A checksum is a number computed using a simple arithmetic formula that can help you determine if the message has been garbled. Here is an example:  $Checksum = 255 - \text{Sum of all data bytes in message}$ .
  - On the receiving end you can check that the length and checksum for the message you received match the actual message. If they don't, you can decide to discard the message or request new data from the device.
  - Example: [4] [253] [1] [1] where 4 is the length (includes length and checksum bytes), [1] [1] are the data bytes (meaning depends on what the sensor does), and  $253 = 255 - (1+1)$  is the checksum (does not include itself or length byte).
- *Streaming vs. polling:*

- Polling a sensor is simple. You send it a message requesting some data each time you want to take a measurement; then it returns a message containing that data.
- Streaming means that you send the sensor a message to turn it on. Then it begins sending back measurements as it gets them – usually at a regular interval

### *What You Need to Know Before You Continue*

Go to the computer you will use, your device, and all the documentation that came with it.

- *Find the Serial Port* on the PC. If there is none, use a USB-to-Serial Converter. You may have to install drivers for it. Connect the device to the PC.
- *COM Port Number*: Each serial port on the PC is labeled COM1, COM2, etc. You need to figure out which one you're attached to.
  - If there is a serial port at the back of the computer, chances are it's COM1. However, even if there is only 1 serial port visible in the back, its possible its COM2, or COM3 especially if there is a modem.
  - If you use a converter or blue tooth the port number gets assigned in software and may change each time you disconnect or reconnect the device. On Windows XP (or vista classic view), go to *Control Panel/System/Hardware/DeviceManager/Ports/Comm Ports* and verify which port your device was assigned
- *Device Settings*: Go to the documentation that came with your device and identify the Communication settings. We'll need this for the next section. Here is an example:
  - Baud Rate
  -
- *Serial Command Interface (SCI)*: Go to the documentation that came with your device and find the serial command interface (may have a different name). It explains how messages are formatted and how to interpret the results. We'll need this later

## **Setting up Serial port objects**

### *Basic Concepts*

- Matlab uses a special variable type to keep track of serial connections – the Serial Object.
- Unlike normal variables which have a single value, objects have many "attributes" or parameters that can be set. (ex. port number, baud rate, buffersize, etc)
- One of those attributes is the port number. A label that corresponds to which port your device is connected to.
- In order to actually send or receive data through the serial port object it must be open. When not in use it can be closed (not the same as deleting it)
- You can have many different serial objects in memory. They can all send and receive data at the same time as long as they are each on a different port.
- There can even be several objects associated with the same physical port. However, only one of those objects associated with a given port can actually be open (sending or receiving data) at any time.

### *Creating a Serial Port Object*

Here is an example of how to do this. the only piece of information you must supply is which com port to use. The rest of the attributes are set to some default values:

```
serialPort = serial('com1')
```

```
Serial Port Object : Serial-COM1
```

```
Communication Settings
```

```
Port: COM1
```

```
BaudRate:          9600
Terminator:        'LF'

Communication State
Status:            closed
RecordStatus:      off

Read/Write State
TransferStatus:    idle
BytesAvailable:    0
ValuesReceived:    0
ValuesSent:        0
```

Note that this list of parameters and attributes it returns is not exhaustive.

### *Setting the Parameters*

Most of the time you don't want to use the default values. You can view or change any attribute using the functions `get` and `set`.

```
get(serialPort, 'baudrate')
```

```
ans =
     9600
```

```
set(serialPort, 'BaudRate', 19200)
get(serialPort, 'BaudRate')
```

```
ans =
     19200
```

This method is cumbersome if you have a lot of things you want to change. A better way to to set them when you create the Serial Object.

```
serialPort_new = serial('com1', 'baudrate', 19200, 'terminator', 'CR')
```

```
Serial Port Object : Serial-COM1
```

```
Communication Settings
Port:          COM1
BaudRate:      19200
Terminator:    'CR'

Communication State
Status:        closed
RecordStatus:  off

Read/Write State
TransferStatus: idle
BytesAvailable: 0
ValuesReceived: 0
ValuesSent:    0
```

You can list as many properties as you want. The name of the property goes in single quotes (check spelling) and the value follows (if the value is text then use single quotes)

## The Parameters

To see a list of parameters and their current values

**get(serialPort)**

```
ByteOrder = littleEndian
BytesAvailable = 0
BytesAvailableFcn =
BytesAvailableFcnCount = 48
BytesAvailableFcnMode = terminator
BytesToOutput = 0
ErrorFcn =
InputBufferSize = 512
Name = Serial-COM1
ObjectVisibility = on
OutputBufferSize = 512
OutputEmptyFcn =
RecordDetail = compact
RecordMode = overwrite
RecordName = record.txt
RecordStatus = off
Status = closed
Tag = GarminGPS
Timeout = 0
TimerFcn =
TimerPeriod = 1
TransferStatus = idle
Type = serial
UserData = []
ValuesReceived = 0
ValuesSent = 0
```

SERIAL specific properties:

```
BaudRate = 19200
BreakInterruptFcn =
DataBits = 8
DataTerminalReady = on
FlowControl = none
Parity = none
PinStatus = [1x1 struct]
PinStatusFcn =
Port = COM1
ReadAsyncMode = continuous
RequestToSend = on
StopBits = 1
Terminator = LF
```

Note that some values are just numbers, while others can only take on certain values in a list ( ex. 'on' or 'off '). To see a list of all parameters with valid choices type (note that the curly brace denotes the default value)

**set(serialPort)**

```
ByteOrder: [ {littleEndian} | bigEndian ]
BytesAvailableFcn: string -or- function handle -or- cell array
BytesAvailableFcnCount
BytesAvailableFcnMode: [ {terminator} | byte ]
```

ErrorFcn: string -or- function handle -or- cell array  
InputBufferSize  
Name  
ObjectVisibility: [ {on} | off ]  
OutputBufferSize  
OutputEmptyFcn: string -or- function handle -or- cell array  
RecordDetail: [ {compact} | verbose ]  
RecordMode: [ {overwrite} | append | index ]  
RecordName  
Tag  
Timeout  
TimerFcn: string -or- function handle -or- cell array  
TimerPeriod  
UserData

SERIAL specific properties:  
BaudRate  
BreakInterruptFcn: string -or- function handle -or- cell array  
DataBits  
DataTerminalReady: [ {on} | off ]  
FlowControl: [ {none} | hardware | software ]  
Parity: [ {none} | odd | even | mark | space ]  
PinStatusFcn: string -or- function handle -or- cell array  
Port  
ReadAsyncMode: [ {continuous} | manual ]  
RequestToSend: [ {on} | off ]  
StopBits  
Terminator

### *Suggestions on Parameters*

Some of these you don't really need to change. Others you will want to change.

#### **Always Set**

You always have to set this to match what is specified in the documentation that came with your device.

- *BaudRate*

#### **Always Check**

The defaults here are usually OK, but you should check that they match whatever is specified in the device documentation.

- *Terminator* (sometimes have to change) 'LF' is linefeed, 'CR' is carriage return, etc
- *FlowControl* (defaults usually OK)
- *Parity* (defaults usually OK)
- *DataBits* (defaults usually OK)
- *ByteOrder* (more on this later)

#### **Good Idea To Set**

Your device will work without setting these but you can set these to make your life easy later.

- *Tag*: The tag is like giving the serial port object a nickname. If have a few different serial ports open this a good way to keep track of them. Example, serialPort is configured to talk with a garmin GPS.  
`set(serialPort, 'tag', 'GarminGPS')`
- *Timeout*: If you try to read data from the serial port and there is no data in the buffer matlab will keep trying to read for "Timeout" seconds (default 10 sec):  
`get(serialPort, 'Timeout')`

```
ans =  
    10
```

This might really slow down your code. There are ways around this, but if there is no data there you probably don't want to sit there for 10 seconds, so consider making it smaller. On the other hand, it does take some time for messages to pass over the wire, so setting it to zero means you will probably miss a lot of messages.

- *InputBufferSize*: This specifies how long the buffer is. The default is 512 bytes. That might not be long enough for your messages. Especially if you think the sensor will be streaming data back more frequently than you plan on reading the buffer. Remember if the sensor tries to send data and the buffer is full it will discard some old data and it will be gone forever. On the otherhand, having an unnessecarily large buffer can be cumbersome.

## Closing Serial Port Objects

### Concepts

When your done with a serial port object it doesn't go away. Also, closing it, deleting it from memory and clearing it from the workspace are three separate actions.

### Example Code

For technical reasons you have to use this systax to properly get rid of it:

```
delete(serialPort_new)  
clear serialPort_new
```

## Comprehensive Example On Creating a New Port

Sometimes, if your program does not terminate correctly you have to abort (CTRL-C) before you can properly delete or close a port. So, it is good practice to check for old serial objects before creating new ones:

```
oldSerial = instrfind('Port', 'COM1'); % Check to see if there are  
existing serial objects (instrfind) whos 'Port' property is set to  
'COM1'  
% can also use instrfind() with no input arguments to find ALL existing  
serial objects  
if (~isempty(oldSerial)) % if the set of such objects is not(~) empty  
    disp('WARNING: COM1 in use. Closing.')    delete(oldSerial)  
end  
  
% creating a new serial object for my GPS (note I can do all this in one  
line if I wanted to  
serGPS = serial('COM1'); % Define serial port  
set(serGPS, 'BaudRate', 4800); % instructions for GPS gave me this  
set(serGPS, 'Tag', 'GPS'); % give it a name for my own reference
```



```
set(serGPS, 'Timeout', .1); %I am willing to wait 0.1 secs for data to  
arrive  
% I wanted to make my buffer only big enough to store one message  
set(serGPS, 'InputBufferSize', 390 )  
get(serGPS) %so you can see my result
```

```
WARNING: COM1 in use. Closing.  
  ByteOrder = littleEndian  
  BytesAvailable = 0  
  BytesAvailableFcn =  
  BytesAvailableFcnCount = 48  
  BytesAvailableFcnMode = terminator  
  BytesToOutput = 0  
  ErrorFcn =  
  InputBufferSize = 390  
  Name = Serial-COM1  
  ObjectVisibility = on  
  OutputBufferSize = 512  
  OutputEmptyFcn =  
  RecordDetail = compact  
  RecordMode = overwrite  
  RecordName = record.txt  
  RecordStatus = off  
  Status = closed  
  Tag = GPS  
  Timeout = 0.1  
  TimerFcn =  
  TimerPeriod = 1  
  TransferStatus = idle  
  Type = serial  
  UserData = []  
  ValuesReceived = 0  
  ValuesSent = 0
```

```
SERIAL specific properties:  
BaudRate = 4800  
BreakInterruptFcn =  
DataBits = 8  
DataTerminalReady = on  
FlowControl = none  
Parity = none  
PinStatus = [1x1 struct]  
PinStatusFcn =  
Port = COM1  
ReadAsyncMode = continuous  
RequestToSend = on  
StopBits = 1  
Terminator = LF
```

## Writing To The Serial Port

Before you can write to your serial port, you need to open it:  
**fopen(serGPS)**

Now you need to figure out two things from the Serial Command Interface (SCI) that came with your device:

1. Will you send binary data (bytes) or text (ascii)?
2. What will you send to it?

If your SCIs messages look like a list of numbers (ex: [4][253][1][1]), its probably the first choice. Note that even though what you send is actually binary, the documentation might list it as numbers between 0 and 255, or hexadecimal numbers.

If your SCIs messages look like a mix of text and numbers (ex: 'MOVE 31'), its probably the second choice.

### *Writing Binary Data*

Use the command fwrite to send four bytes of binary data  
fwrite(serGPS, [0, 12, 117, 251]);

### *Writing ASCII Commands*

Use the command fprintf to send ascii data. You can use a mix of text in single quotes and variables values.

```
moveNum = 98; pauseTime = 2; % just some example data
fprintf(serGPS, 'MOVE %d, PAUSE %d', [moveNum, pauseTime] ); % note
```

Its important to understand that a number, (ex. 98) is not sent as a number. Its actually the ascii code for the characters '9' and '8'.

### *Example Code*

## **Reading From The Serial Port**

### **Streaming vs Polling: Flushing the Buffer**

If you are going to poll the device (send it a request each time you want to get data) you don't want to read any old data that might be left over in the buffer. This is a useful and quick way to clean it out

```
N = serRoomba.BytesAvailable();
while(N~=0)
    fread(serRoomba,N);
    N = serRoomba.BytesAvailable();
end
```

## **Reading Formatted ASCII**

Say my device returns a text sentence like this

```
X, 2.1, Y, 3.2, T, -0.5
```

Where X is the x position, Y the y position and T is the heading.

```
sentence = fscanf(serialObj, '%s'); % this reads in as a string (until
a terminator is reached)
[x, xPosition, y, yPosition, t, Heading] = streadd(sentence,
'%s%f%s%f%s%f', 1, 'delimiter', ',')
% decodes "sentence" as string, float, string, float, string, float"
seperated (delimted) by commas

ans =

x = 'X'
xPosition = 2.1
y = 'Y'
yPosition = 3.2
t = 'T'
Heading = -0.5
```

## Reading Data

You can use `fread` to read in data (not text). It can automatically format the data for you. Here is an example. Say the buffer currently has 2 bytes of data in it

```
[1], [8]
```

```
a = fread(serialObj, 2);
% Will read two bytes and create a vector
a = [1; 8]
```

If you omit the ,2 the result would be the same -- it will just read until either a terminator is reached or there is no more data in the buffer

Alternatively, suppose you know those two bytes are used to express a singel 16-bit integer, you can use

```
a = fread(serialObj, 1, 'int16')
```

```
ans =
```

```
264
```

Which is equivalent to  $1*(256^1) + 8*(256^0)$

Note that even though we only "read" 1 value, two elements were take out of the buffer since a 16 bit interger is actually composed of two bytes. To see a list of all format types, type

```
>> help serial/fread
```

## Putting It All Together

### Good Programming Advice

- to make your code crash resistance try: preinitializing any return variables, then put the reading and writing to the serial port inside a TRY-CATCH statement
- Debugging??

## Example 1: GPS Initialization

```
function [serGPS] = initializeGarmin(ComPortNumber)
%This function initializes the serial port properly
% for use with the Garmin GPS
%COMPORT is the number of the serial port:  ex use 3 for 'COM3'
% port number can be checked in
% Control Panel/System/Hardware/DeviceManager/Ports
% serGPS output is a matlab serial port object
% Esposito 6/25/2008 with code from regneier, bishop, et al; modified by
% MIDN 1/C Li
```

```
port = strcat('COM',num2str(ComPortNumber) );
```

```
out = instrfind('Port', port); % Check to see if THAT serial port is
already defined in MATLAB
if (~isempty(out)) % It is
    disp('WARNING: port in use. Closing.')
    if (~strcmp(get(out(1), 'Status'),'open')) % Is it open?
        delete(out(1)); % If not, delete
    else % is open
        fclose(out(1));
        delete(out(1));
    end
end
end
```

```
serGPS = serial(port); % Define serial port
set(serGPS, 'BaudRate', 19200); % Default Baud rate of 19200
set(serGPS, 'Tag', 'GPS'); % give it a name
set(serGPS, 'TimeOut', .1);
% want to make the buffer big enough that new message can always fit
%(example is 389 characters long but message length is variable)
% but not so big as to hold 2 messages
set(serGPS, 'OutputBufferSize', floor(389*1.5) );
set(serGPS, 'InputBufferSize', floor(389*1.5) );
fopen(serGPS); % Open it;
pause(1) % give it a second to start getting data
disp('Garmin Initialized')
```

### Example 1: Read GPS

```
function [lat, lon] = ReadGarmin(serGPS)
% Reads a streaming GPS.  If no data, does not wait, returns nans.
% Note that this code never crashes.  Even if the GPS unit dies or
gets unplugged, or cant find satelites.

% serial port must first be initialized using initializeGarmin
% inputs: serGPS (from initializeGarmin)
% Outputs: x (lon) and y(lat)

%% initialize to nan, will have something to return even if serial
comms fail
lat =nan; lon = nan;
```

```
%% IF THERE IS NO DATA?
if (get(serGPS, 'BytesAvailable')==0)
    disp('Data not avail yet.  Try again or check transmitter.')
    return
end

%% IF THERE IS DATA
while (get(serGPS, 'BytesAvailable')~=0)
    try
        % read until terminator
        sentence = fscanf(serGPS, '%s');
        Ns = length(sentence);

        % Make sure header is there
        if strcmp(sentence(1:6), '$GPRMC')

[prefixRMC,timeRMC,ActiveRMC,lat,latdirRMC,lon,londirRMC,spdKnots,Angled
eg] = strread(sentence, '%s%f%s%f%s%f%s%f%f', 1, 'delimiter', ',');

        % these cases mean that the GPS can't find satelities
        % (status not equal to A (active)
        % or the sentence wasn't long enough to fill in lat and lon
            if isempty(lat)|| (ActiveRMC{1} ~= 'A')
                lat = nan;
            end
            if isempty(lon)|| (ActiveRMC{1} ~= 'A')
                lon = nan;
            end
        end
    end

catch ERR_MSG
    % if something didn't work correctly the error message displays
    disp('Error Reading Data! Check Unit')

end

end
```

## Example 2: Initialize the iRobot Create

```
function [serPort] = RoombaInit(my_COM);
% initializes serial port for use with Roomba
% COMmport is the number of the comm port
% ex. RoombaInit(1) sets port = 'COM1'
% note that it sets baudrate to a default of 57600
% can be changed (see SCI). Will NOT work if robot is plugged into
% charger.
% An optional time delay can be added after all commands
% if your code crashes frequently
global td
td = 0.01;
% This code puts the robot in SAFE(130) mode, which means robot stops
% when cliff sensors or wheel drops are true; or when plugged into
charger
Contrl = 132;

% Esposito 9/2008

warning off

%% set up serial comms,
% output buffer must be big enough to take largest message size
comm = strcat('COM', num2str(my_COM));

a = instrfind('port',comm);
if ~isempty(a)
    disp('That com port is in use. Closing it.')
    fclose(a)
    delete(a)
end

disp('Establishing connection to Roomba...');

% defaults at 57600, can change
serPort = serial(comm,'BaudRate', 57600);
set(serPort,'Terminator','LF')
set(serPort,'InputBufferSize',100)
set(serPort, 'Timeout', 1)
set(serPort, 'ByteOrder','bigEndian');
set(serPort, 'Tag', 'Roomba')

disp('Opening connection to Roomba...');
fopen(serPort);

%% Confirm two way communication
disp('Setting Roomba to Control Mode...');
% Start! and see if its alive
Start=[128];
fwrite(serPort,Start);
pause(.1)
```

```
fwrite(serPort,Contrl);
pause(.1)
% light LEDS
fwrite(serPort,[139 25 0 128]);

% set song
fwrite(serPort, [140 1 1 48 20]);
pause(0.05)
% sing it
fwrite(serPort, [141 1])

disp('I am alive if my two outboard lights came on')

confirmation = (fread(serPort,4))
pause(.1)
```

## Example 2: Read all sensors from the Create.

```
function [BumpRight, BumpLeft, BumpFront, Wall, virtWall, CliffLft, ...
    CliffRgt, CliffFrntLft, CliffFrntRgt, LeftCurrOver,
    RightCurrOver, ...
    DirtL, DirtR, ButtonPlay, ButtonAdv, Dist, Angle, ...
    Volts, Current, Temp, Charge, Capacity, pCharge] =
    AllSensorsReadRoomba(serPort);
% Reads all 23 Roomba Sensors from a single data packet.  Values are
% [BumpRight (0/1), BumpLeft(0/1), BumpFront(0/1), Wall(0/1),
virtWall(0/1), CliffLft(0/1), ...
%   CliffRgt(0/1), CliffFrntLft(0/1), CliffFrntRgt(0/1), LeftCurrOver
(0/1), RightCurrOver(0/1), ...
%   DirtL(0/1), DirtR(0/1), ButtonPlay(0/1), ButtonAdv(0/1), Dist
(meters since last call), Angle (rad since last call), ...
%   Volts (V), Current (Amps), Temp (celcius), Charge (milliamphours),
Capacity (milliamphours), pCharge (percent)]
% Can add others if you like, see code
% Esposito 3/2008
warning off
global td
sensorPacket = [];
% flushing buffer
confirmation = (fread(serPort,1));
while ~isempty(confirmation)
    confirmation = (fread(serPort,26));
end

%% Get (142) ALL(0) data fields
fwrite(serPort, [142 0]);

%% Read data fields
BmpWheDrps = dec2bin(fread(serPort, 1),8); %

BumpRight = bin2dec(BmpWheDrps(end)) % 0 no bump, 1 bump
```

```
BumpLeft = bin2dec(BmpWheDrps(end-1))
if BumpRight*BumpLeft==1 % center bump sensor is really just
% left AND right at same time
    BumpRight =0;
    BumpLeft = 0;
    BumpFront =1;
else
    BumpFront = 0;
end
Wall = fread(serPort, 1) %0 no wall, 1 wall

CliffLft = fread(serPort, 1) % no cliff, 1 cliff
CliffFrntLft = fread(serPort, 1)
CliffFrntRgt = fread(serPort, 1)
CliffRgt = fread(serPort, 1)

virtWall = fread(serPort, 1)%0 no wall, 1 wall

motorCurr = dec2bin( fread(serPort, 1),8 );
Low1 = motorCurr(end); % 0 no over current, 1 over Current
Low0 = motorCurr(end-1); % 0 no over curr, 1 over Curr
Low2 = motorCurr(end-2); % 0 no over curr, 1 over Curr
LeftCurrOver = motorCurr(end-3) % 0 no over curr, 1 over Curr
RightCurrOver = motorCurr(end-4) % 0 no over curr, 1 over Curr
DirtL = fread(serPort, 1)
DirtR = fread(serPort, 1)

RemoteCode = fread(serPort, 1); % could be used by remote or to
communicate with sendIR command
Buttons = dec2bin(fread(serPort, 1),8);
ButtonPlay = Buttons(end)
ButtonAdv = Buttons(end-2)

Dist = fread(serPort, 1, 'int16')/1000 % convert to Meters, signed,
average dist wheels traveled since last time called...caps at +/-32
Angle = fread(serPort, 1, 'int16')*pi/180 % convert to radians, signed,
since last time called, CCW positive

ChargeState = fread(serPort, 1);
Volts = fread(serPort, 1, 'uint16')/1000
Current = fread(serPort, 1, 'int16')/1000 % neg sourcing, pos charging
Temp = fread(serPort, 1, 'int8')
Charge = fread(serPort, 1, 'uint16') % in mAhours
Capacity = fread(serPort, 1, 'uint16')
pCharge = Charge/Capacity *100 % May be inaccurate
%checksum = fread(serPort, 1)
pause(td)
```